An Overview of BlueGene/L and A Short Tutorial on BGLSim

Bronis R. de Supinski

October 30, 2003

UCRL-PRES-200704

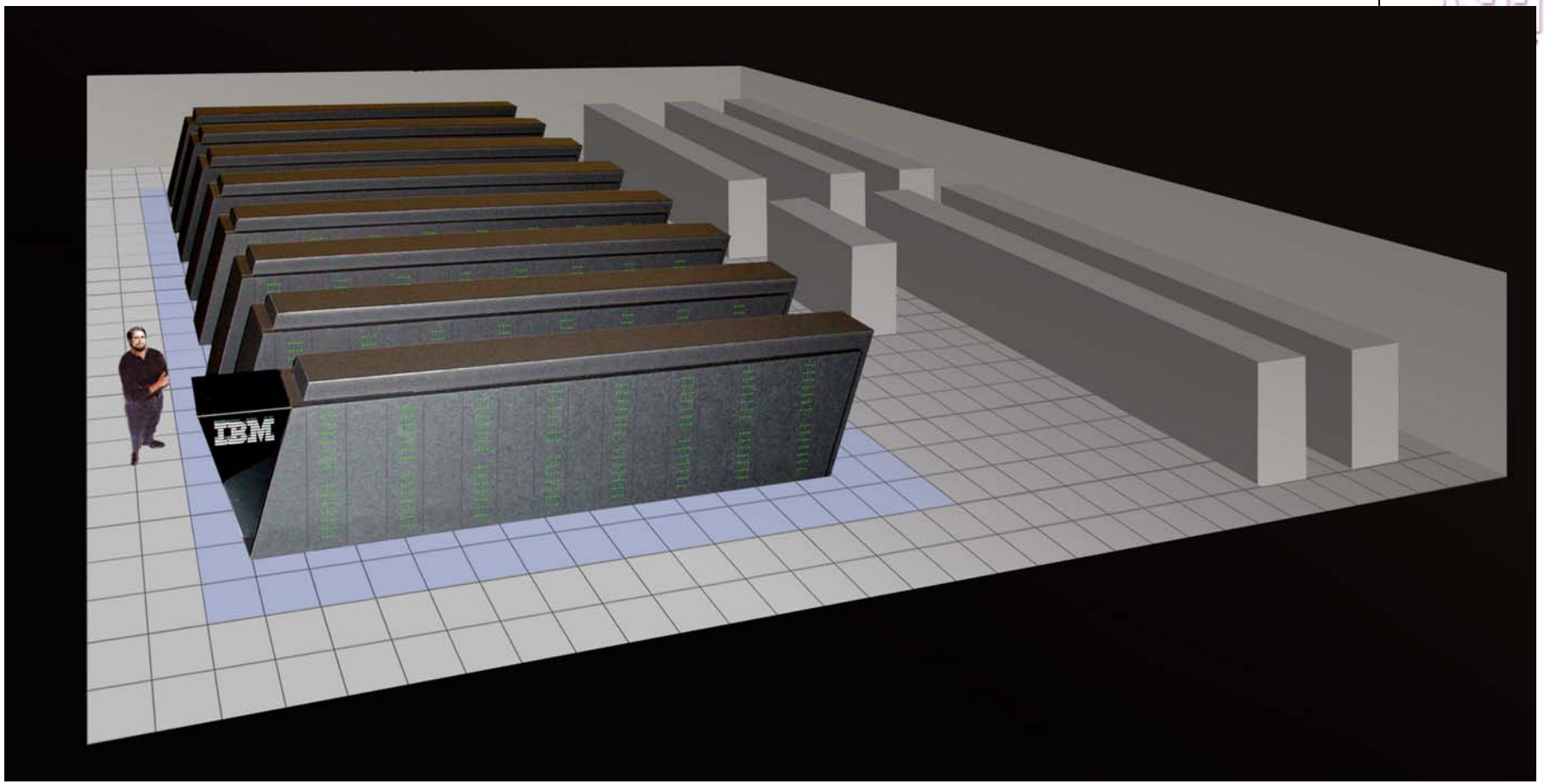
Note: This presentation borrows heavily from presentations prepared by several others

BlueGene/L is an architecture optimized for cost, performance and scalability

- Partnership between IBM and DOE/NNSA, LLNL lead lab
  - Address a limited but large set of applications
- Low power, low cost, high performance: *180-360 Tflops*
  - New generation of embedded processors
  - Large on chip DRAM
  - High speed, low latency, low power serial links
  - Simplified OS
- Attacks the distance to memory problem
  - Low latency memory
  - High bandwidth memory
- Significant savings in facilities cost
- Scalable to petaFLOP/s systems

UCRL-PRES-147124

This artist concept for 360 Tflops BlueGene/L illustrates its remarkably compact footprint

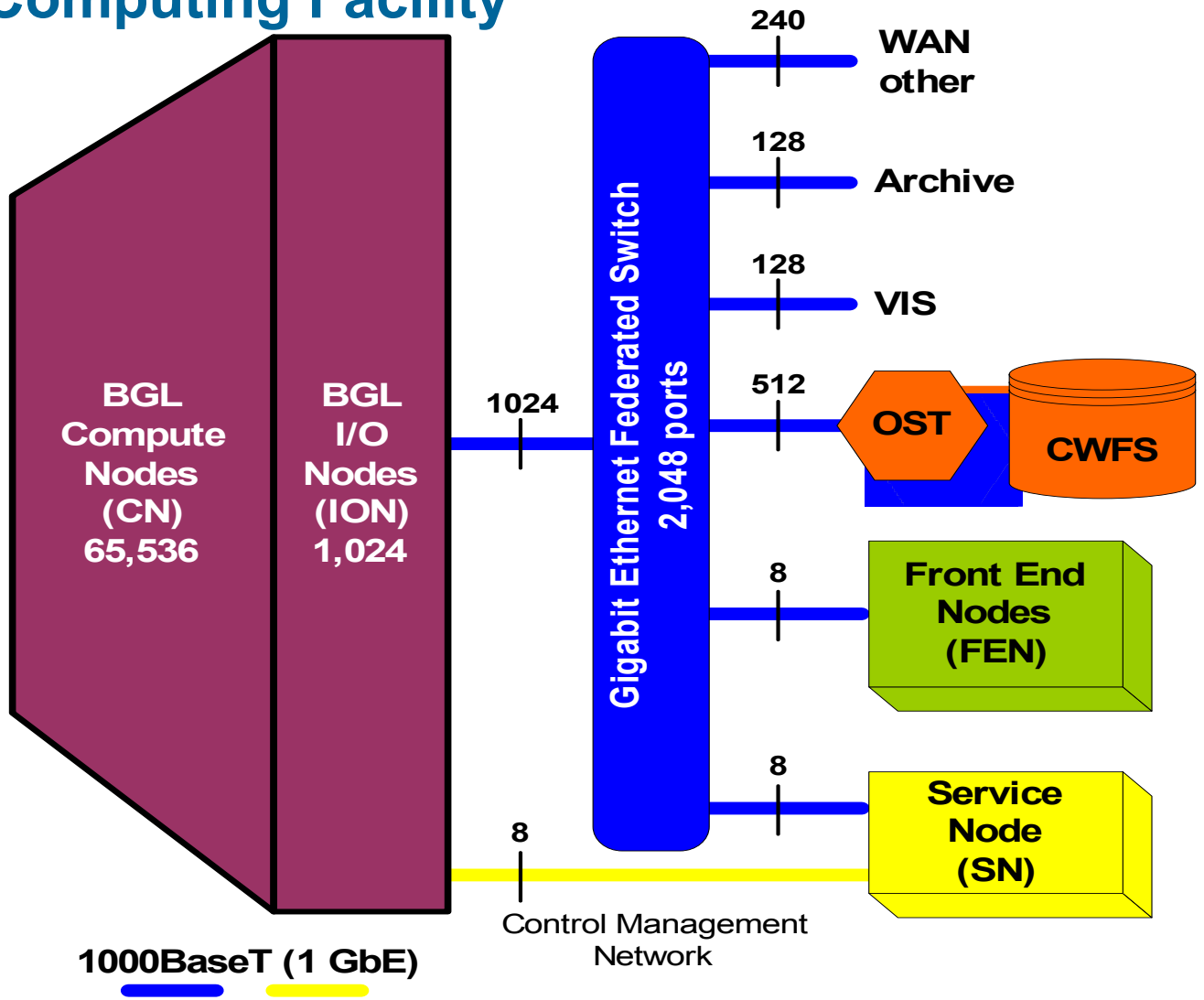


2,500 ft^2 footprint includes 400 TB of disk storage

LLNL will support a wide array of application teams on BlueGene/L

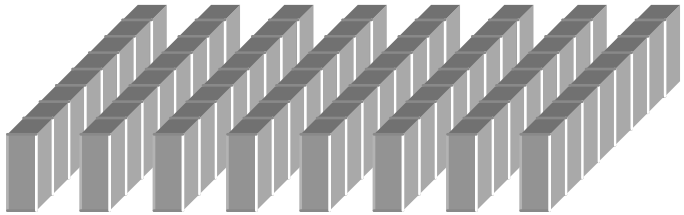
- Currently planned LLNL earliest adopter applications
  - GP
  - DD3d
  - ALE3D
  - Miranda
  - Raptor
- Purple benchmark codes
- ASCI Alliance Centers
- LBNL, ANL, others?

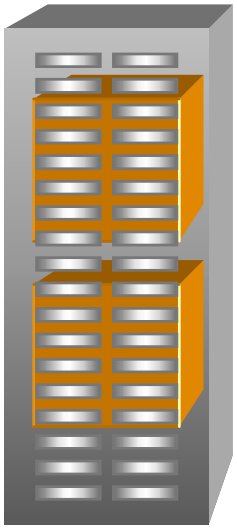
BlueGene/L is stimulating the development of a scalable storage area network for LLNL's Open Computing Facility

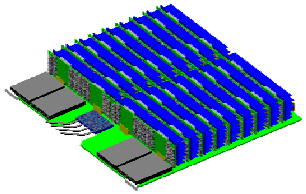


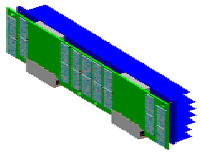
The high-level of integration results in a compact footprint

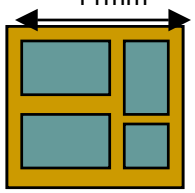
Building BlueGene/L











Compute Chip

2 processors
2.8/5.6 GF/s
4 MiB* eDRAM

Compute Card

FRU 25mmx32mm
2 compute chips
(2x1x1)
2.8/5.6 GF/s
256 MiB* DDR
15 W

Node Card

32 compute chips
16 compute cards
(4x4x2)
90/180 GF/s
8 GiB* DDR

Midplane

SU (scalable unit)
16 node boards
(8x8x8)
1.4/2.9 TF/s
128 GiB* DDR
7-10 kW

Cabinet

32 node boards
(8x8x16)
2.9/5.7 TF/s
256 GiB* DDR
15-20 kW

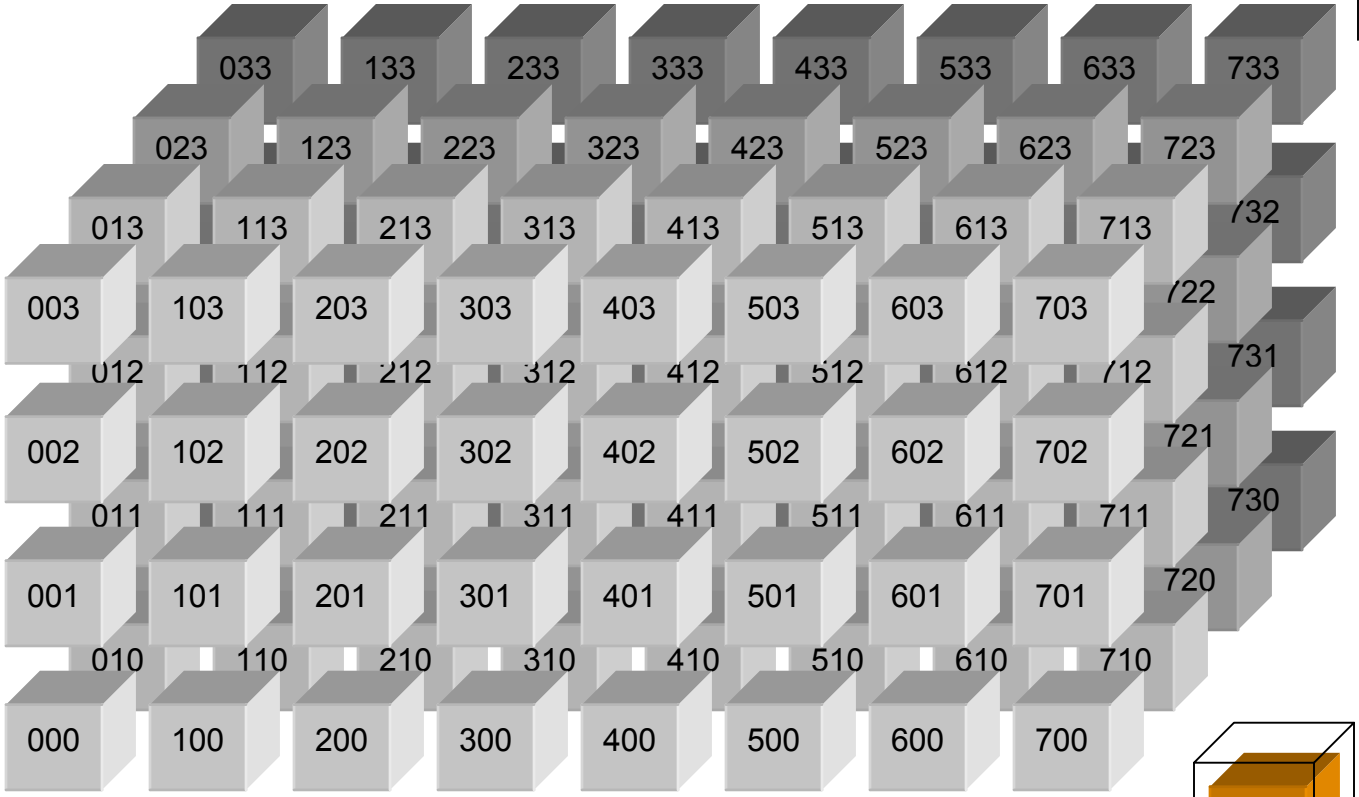
System

64 cabinets
(32x32x64)
180/360 TF/s
16 TiB*
~1 MW
2500 sq.ft.

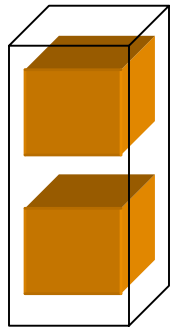
(compare this with a 1988 Cray YMP/8 at 2.7 GF/s)

* http://physics.nist.gov/cuu/Units/binary.html

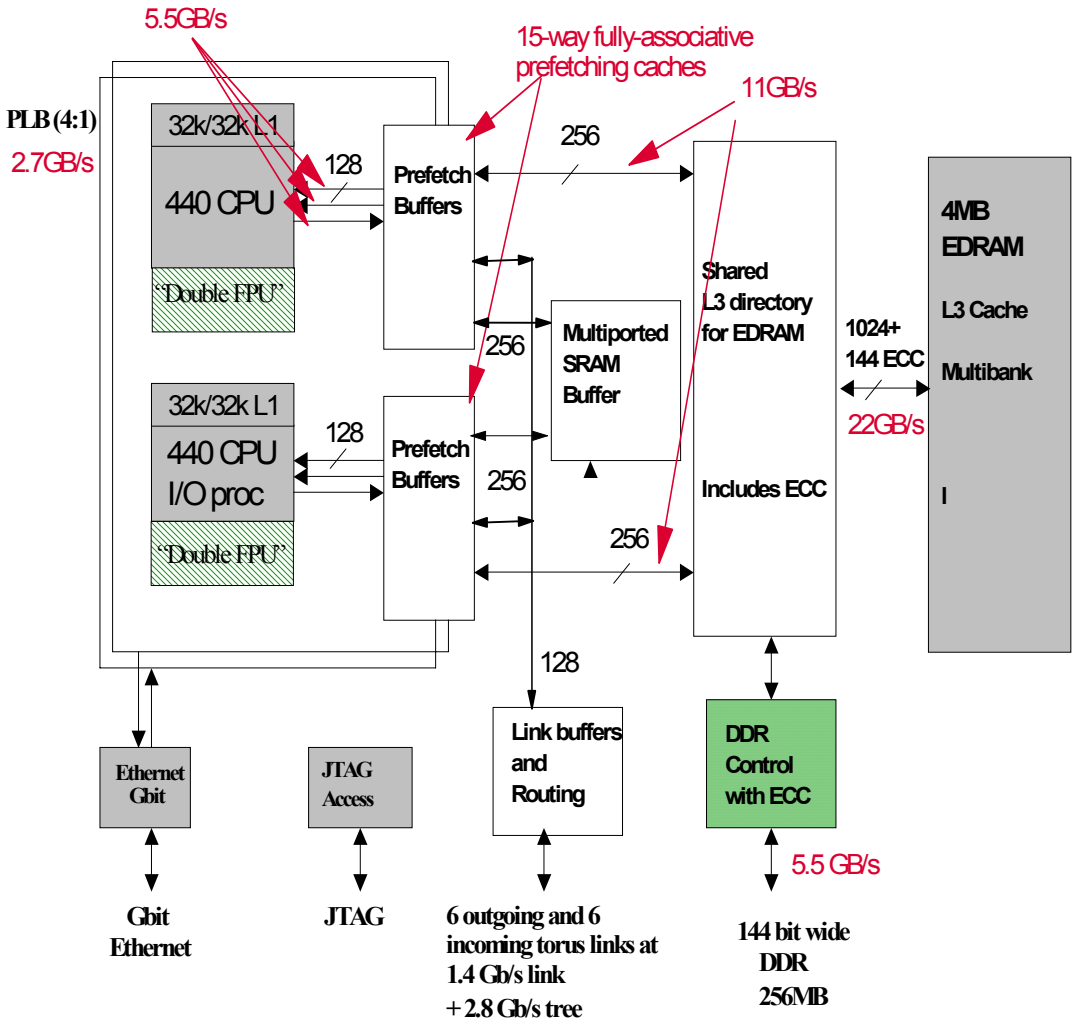
A midplane contains 512 nodes (8x8x8, 2.9TF/s) and is the scalable unit, either connected or isolated from neighbors



2 midplanes per cabinet



Each ~15W BlueGene/L compute node is composed of a single ASIC and 9 SDRAM-DDR memory chips – that's it!



The **BlueGene/L** compute ASIC uses IBM CMOS CU-11 0.13µm technology. In this diagram, the ***gray blocks*** are standard System-On-A-Chip offerings from IBM's ASIC library. The *white blocks* require a new design effort, while the ***green blocks*** are developed from existing designs.

All this on a ~11mmx11mm Si die.

PPC440 processor core

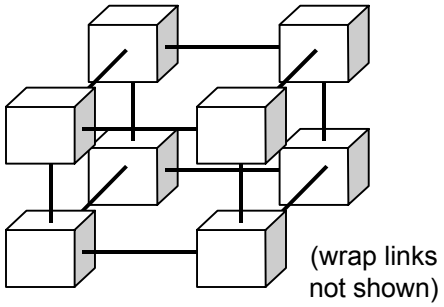
- Target 700 MHz
- 2-way superscalar (2 instructions per cycle)
  - 1.4 Ginstrs per processor
  - 2.8 Ginstrs per node
  - 184 Tinstrs overall
  - ≤6 ops per proc per cycle (e.g., 2 FP mac, 1 int mac)
- 3 execution pipelines:
  - Load/store
    - Up to 3 loads pending
  - Simple integer
  - Complex integer, branch
- 32 32-bit integer registers

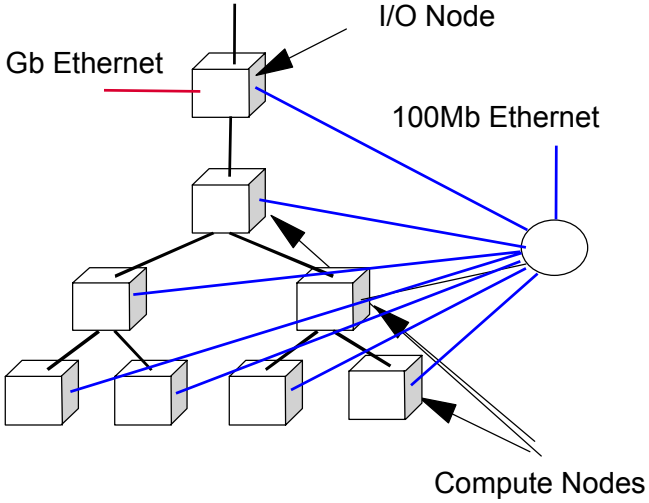
- "Double hummer" FPU added
- Dynamic/static branch prediction
  - 2-bit branch history
  - 1-cycle branch latency
- Dedicated HW loop counter and special loop branch instruction

Floating point "Double Hummer"

- 2 64-bit FPUs per core (so 4 per node)
- 32 64-bit register *pairs* service the 2 FPUs
- An instruction can drive *either* FPU or *both* (SIMD)
  - A SIMD Multiply-Accumulate does 4 64-bit Flops
  - Also has complex, other intra-pair instructions
- 2 FMAs × 1 core × 64k nodes @ 700 MHz = 184 Tflops peak
  - Using 2nd core's DH-FPU gives 367 Tflops peak
  - Using single-op instrs (non-MAC) reduces by 1/2
  - To approach peak, avoid reading off-chip memory
    - Load BW from L3: 64 bits every .25 cycles
    - Load BW from memory: 64 bits every 1.4 cycles
- Quadword load fills a register pair (also useful for comm)

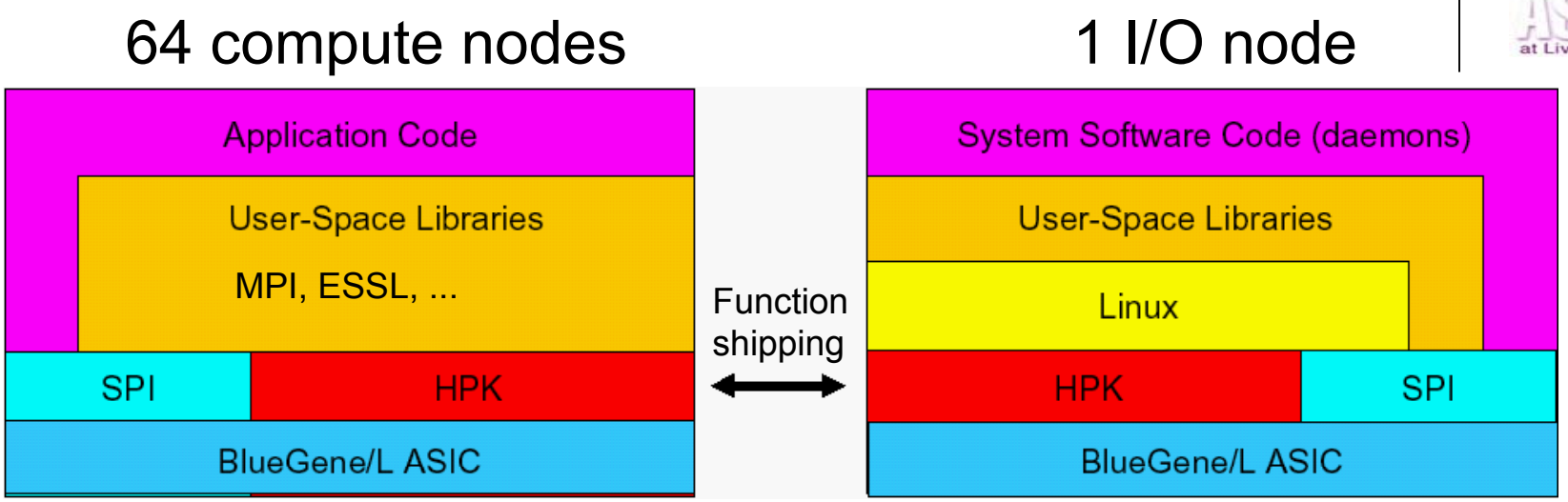
Architectural features of BlueGene/L promote application efficiency and scaling – nodes connected by 5 networks





- 3D torus for point-to point messages
  - 6 bi-directional nearest-neighbor links
  - 4.2GB/s target aggregate bandwidth
  - Support for broadcast operations
- Binary combining tree
  - Target latency of about 2 μs
  - Supports 32 bit integer and logical ops
  - Software extensions for floating point
- JTAG for diagnostics and IPL
  - Allows access to the processor's registers
  - Connected to the 100 Mbit Ethernet port
- Gigabit ethernet to I/O nodes
  - Tree connects I/O node to 64 Compute nodes
- Low-latency global barrier network

BlueGene/L software overview



- Usual mode: 1 MPI process per node
  - 2nd core does communication
- Options for computing on both CPUs
  - Virtual node mode?
  - Co_start pthread-like interface
- High Performance Kernel (HPK) provides basic OS functionality
- No paging, static linking only

- Most Linux services (e.g., files, sockets, IP, NFS) via function shipping
- Linux and user-space code do not directly access hardware
- Supporting C, C++, F95
- UPC, Co-array Fortran, Charm++?
- I/O nodes also serve as portal to outside

Characteristics of BlueGene/L

† target specifications

* comm. co-processor mode / symmetric mode

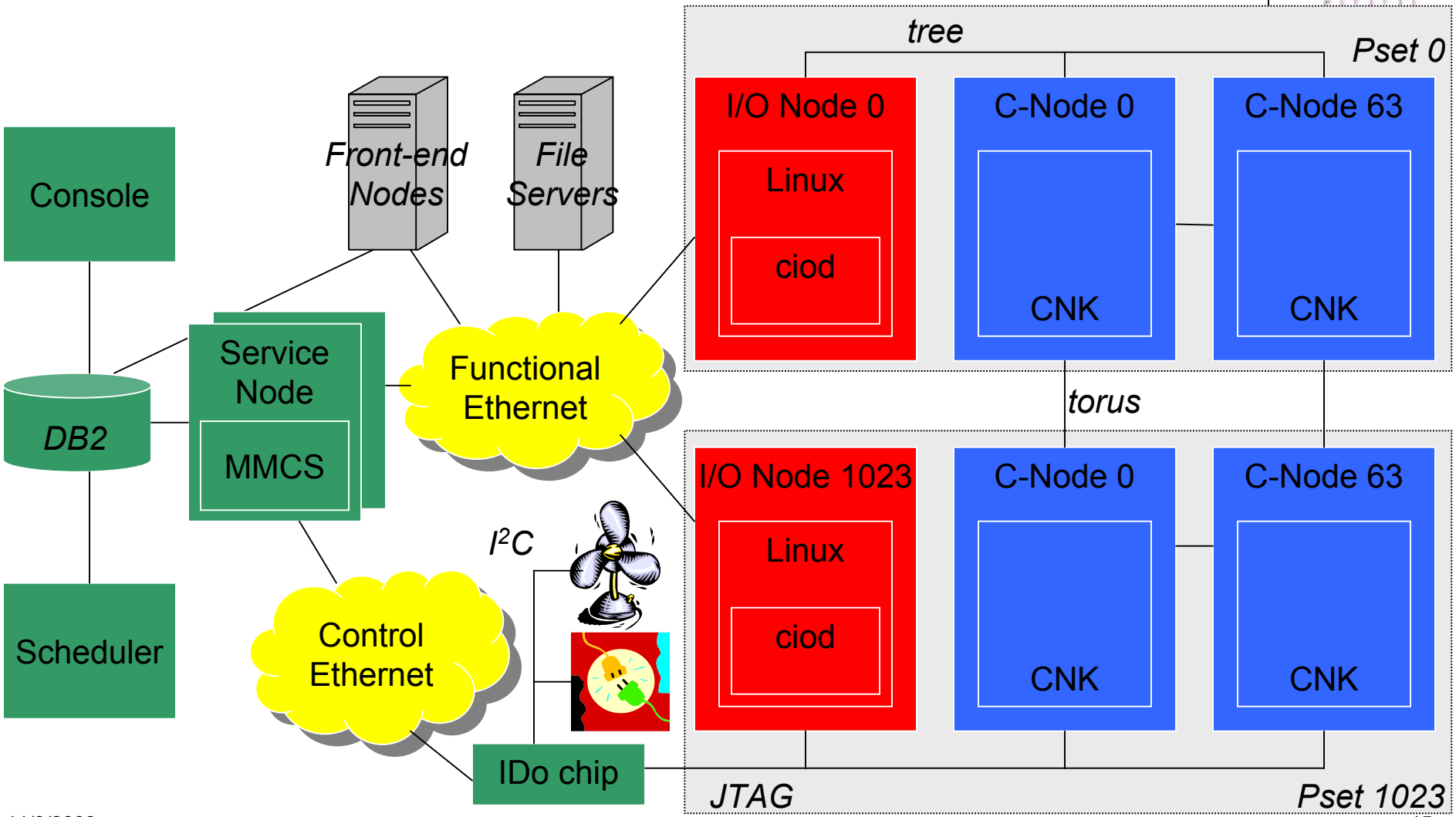
	ASCI White	ASCI Q	Earth Simulator	ASCI Purple	BlueGene/L†
Machine Peak Speed (Tflop/s)	12.3	20	40	100	180 / 360*
Total Memory (Tbytes)	8	22	10	50	16–32
Footprint (ft.^{2})	10,000	20,000	34,000	12,000	2,500
Total Power (MW)	1.0	3.8	10	4.5	1.2
Cost (M$)	~100	~200	~350	~250	<< 100
Installation Date	9/2000	~9/2002	2/2002	12/2004	~12/2004
No. of Nodes	512	4,096	640	197	65,536
CPUs per Node	16	4	8	64	2
Clock Frequency (MHz)	375	1,000	500	~2,000	700
Power Dissipation/Node (W)	2,000	920	16,000	23,000	15
Peak Speed/Node (Gflop/s)	24.0	7.3	64.0	512	2.8
Memory/Node (GiB)	16	8	16	250	0.25–0.5
Memory Bandwidth (TB/s)	8	19	160	130	360
Memory Latency (cycles)	140	330	–	280	70
MPI Latency (μs)	25	4.5	6–20	5-10	7
Interconnect Bandwidth (B:F)	0.042	0.085	0.13	0.13	0.75
Bi-Section Bandwidth (B:F)	0.04	0.04	0.03	0.06	0.008

BlueGene/L's characteristics suggest new metrics to emphasize its dramatic departure from recent supercomputers

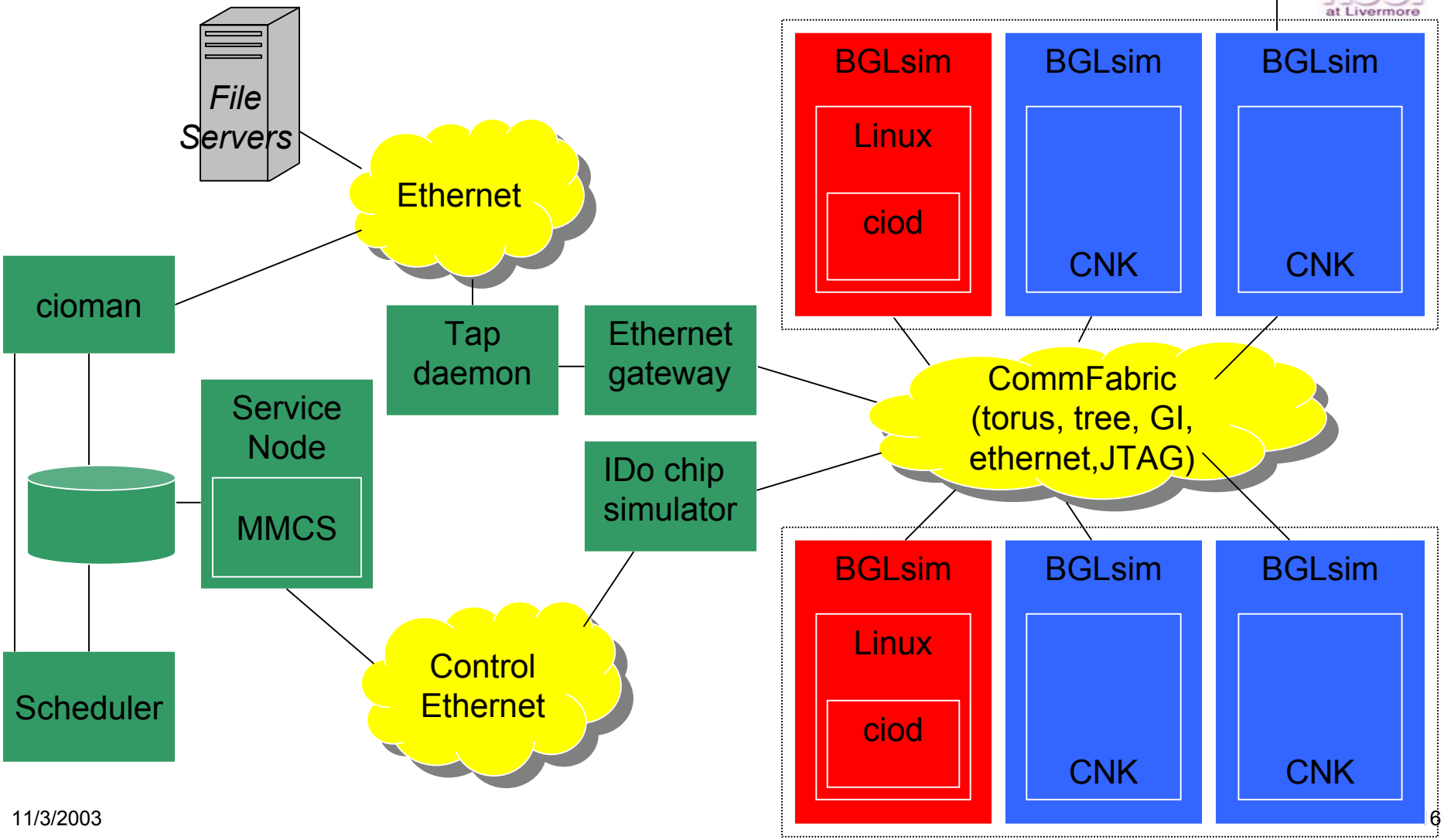
	ASCI White	ASCI Q	Earth Simulator	BlueGene/L†
Memory-Space Effectiveness (GiB/m^2)	8.6	17	3.1	140
Speed-Space Effectiveness (GF/s/m^2)	13	16	13	1600
Speed-Power Effectiveness (GF/s/kW)	12	7.9	4.0	300
Speed-Cost Effectiveness (GF/s/M$)	~100	~100	~100	>> 2,000

† target specifications

Blue Gene/L system software architecture provides a full end-to-end solution



The Blue Gene/L simulation environment models the complete BlueGene/L system software architecture



Overview of BGLSim

- Architectural simulator of a single BG/L node
  - Consumes PPC440 binaries
  - One cycle per instruction
  - Statistics as instruction histograms, traces; timing model
  - Runs on Linux/x86 workstations
- BG/L specific features:
  - Supports 2 PPC 440 cores per chip, 440GP instruction set
  - Hummer^2 (Oedipus ISA) floating point
  - Architecture accurate caches: L1, L2, L3
  - EMAC4, MAL (1Gb/s Ethernet)
  - BG/L interrupt controller (BIC)
  - Torus, tree devices and other networks

Invoking `BGLSim` in single chip mode

`mambo [options]`

- Verbose mode (**`-v`**): print every instruction executed
- Verbose interrupts (**`-z`**): print every interrupt
- Single/dual core mode (**`-S,-D`**)
- Cache model (**`-L:123,12,13,None`**)
- PseudoUART console (**`-x`**): interactive console under Linux
- Interactive mode (**`-i`**): CTRL-C suspends the simulator
  - Peek, poke memory, registers, TLBs
- Preload ELF images (**`-e`**)
  - Significantly faster than loading them through JTAG
- Torus/tree cheat (**`-t`**)
  - Preconfigure torus and tree

Multichip simulation architecture:
BGLMachine

- Machine description file:
  - Racks, midplanes, node cards, compute & I/O cards, ***wiring***
  - Described in XML format
- Used by the real control system and the simulator
- In real hardware:
  - Backed by MMCS database description of same items
  - Generated from the database
- In simulator:
  - Generated when simulation starts
  - Library accessible to simulation components, esp. **`CommFabric`** and **`simboot`**

Multichip simulation architecture: `simboot`

- "Creates" and IPLs a simulated system
  - Creates **`BGLMachine`** file according to arguments
  - Saves **`BGLMachine`** to a file (**`bglsim.xml`**)
  - Creates and saves an MPI (LAM) schema (**`simboot.schema`**)
    - What programs to run where
    - LLNL ported simboot to use Quadrics (and other) MPI libraries
  - Starts the simulator processes
  - IPLs (boots) the simulators
    - "cheating" – (simulators wake up with pre-loaded images)
    - Alternative boot: simulated control system
- Allows simulated architectures not supported in real hardware
  - e.g. 4 compute nodes in a 2x2 torus with 2 I/O nodes
  - Hard-coded configurations
  - Command line arguments to create arbitrary* simulations

Multichip aimulation architecture:
CommFabric

- "Implements" **`BGLMachine`** in simulator
- Simulates cabling and network chips of real hardware
  - All 5 BG/L networks
- **`CommFabric`** is a library linked by all simulation components
- MPI messages
  - Torus: packets routed according to hint bits
  - Tree: packets routed according to class routes
  - Ethernet: packets routed through Ethernet gateway
  - GI: state changes routed through nodes

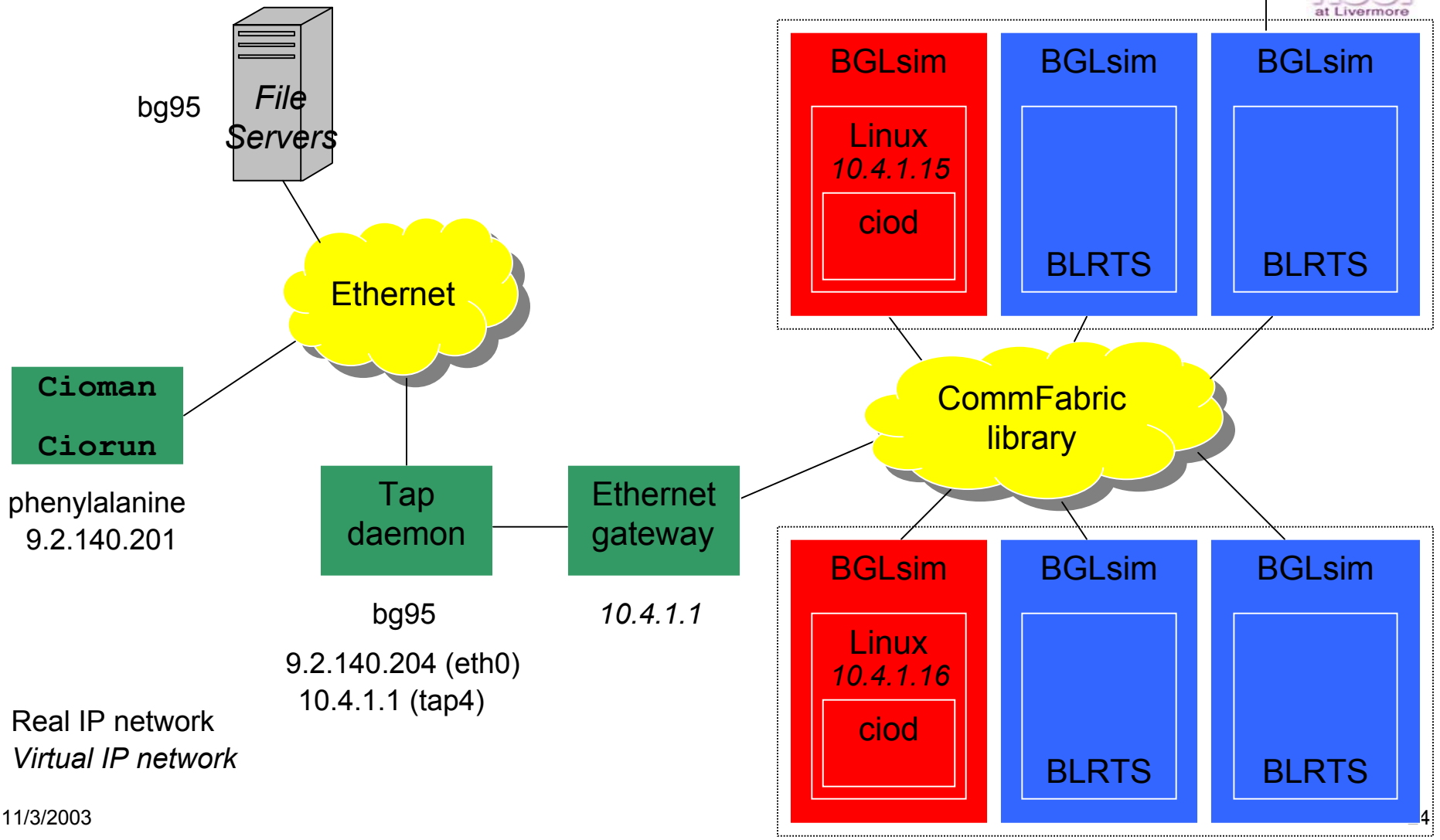
IDo chip simulator, MMCS simulator

- IDo sim: IDo chip & JTAG functional simulator
  - Read, write SRAM
  - Read, write DCRs
  - Apply reset on/off to individual cores
- MMCS_sim: midplane management control system
  - Talks to IDO simulator instead of JTAG network
- MMCS+IDo can boot a simulation
  - simboot starts simulation with all BGLsims running “empty”
  - MMCS loads boot images and resets nodes through IDO sim

TapDaemon and the ethernet gateway

- **bglsim** routes external ethernet packets (not 10.0.0.0) to **TapDaemon** through CommFabric library and Ethernet gateway
  - Internal Ethernet packets routed directly between mambos through CommFabric
- **TapDaemon**
  - Part of simulation: listens for connections on a well known port
    - Hostname and port number defined when installing simulator
  - Requires root privileges because reads/writes raw ethernet
  - Requires recompilation of the host Linux kernel with TUN/TAP module enabled
  - Log in /var/log/tapserver
- Only one **TapDaemon** shared by all simulations
  - As part of initialization, simboot contacts the tapserver, obtains a new simulation number (called netId) and forks a new tap daemon for the simulation (new functionality)
- Ethernet gateway is the interface between **TapDaemon** and simulation
  - Runs with user privileges
  - Reads and writes CommFabric packets (has MPI rank)
  - Reads and writes from/to socket with forked **TapDaemon**

Routing and NFS in `BGLsim` involves both simulated and real ethernet traffic



BGLSim uses cioman and ciorun, the actual job starters fo BlueGene/L

- cioman and ciorun run outside the simulation
- Connect to I/O nodes using CIO protocol
  - over real+simulated ethernet
- Once simulation is booted, anybody can connect to it
  - bglsim.xml describes IP addresses of I/O nodes
- cioman is interactive, allows user process debugging
- ciorun is similar to mpirun, and has a –np argument

Accessing `BGLsim` on LLNL's ASCI Linux Cluster

- The value of experience:
  - Always start with a fresh xterm and ssh to alc
  - Use the makefile structure ("`include Make.rules`")
    - Provides consistent mechanism
    - Supports future upgrades
  - Break simulation into pieces (don't use "`make test`")
  - If things go wrong, start over with a fresh xterm
- Start with the basic examples
  - `cp -r /BlueLight/current/examples .`
  - Include `multichip/Make.rules` in your makefiles…

Trying `BGLsim` examples

- Single node examples show how to build and run:
  - On a bare-bones compute node: `hello_standalone`
  - On a compute node: `hello_blrts`
  - On an I/O node: `hello_linux`
  - We'll focus on multichip examples
- Multichip examples cover running on multiple nodes
  - Simple hello world from multiple compute nodes: `hello`
  - File I/O from compute nodes: `fileio`
  - Low-level communication interface tests: `torus`, `torus2`, `vtorus`, `commworld`, `matmult`
  - A variety of MPI tests
  - We'll cover `hello` and the MPI test `simple`

Running the hello example

- Set the `SIMBOOT_MACHINES` environment variable
  - `setenv SIMBOOT_MACHINES /etc/bgl.hosts`
  - Can really use just about any text file on alc…
- Test the X connection to alc – run "`xclock`"
- `mkdir /bgl/<userid>` (directory to run executables from)
- Build and run the hello example
  - `cd examples/multichip/hello`
  - `more Makefile` (note that `APP=hello.rts`)
  - `make hello.rts`
  - `make start` (starts the simulator, including several xterms)
  - `make run` (runs the simulated hello world program)
- Stop the simulator
  - `squeue` (Locate your simboot jobid `<jobid>`)
  - `scancel <jobid>`

Running the simple example

- Build the simple example
  - `cd ../mpi/simple`
  - `more Makefile` (note that `APP=simple.rts`)
  - `make hello.rts`
- Modify Makefile not to use simulator xterms – add:
  - `XTERMS=`
- Run the simulator and the example
  - `make start` (starts the simulator, without xterms)
  - `make run` (runs the simulated MPI ring program)
- Stop the simulator
  - `squeue` (Locate your simboot jobid `<jobid>`)
  - `scancel <jobid>`

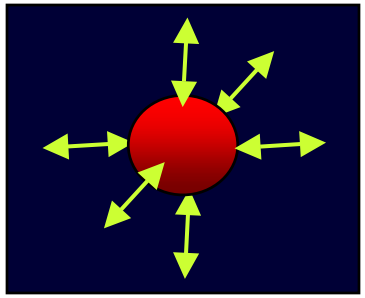
Modifying your code to run in `BGLsim`

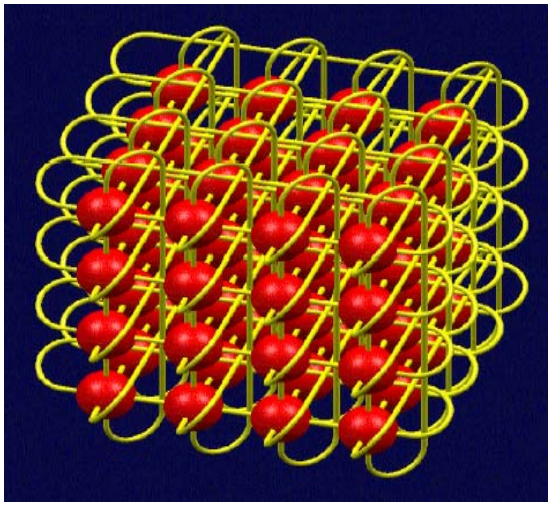
- Add "`include Make.rules`" to your makefile
- Add definition of `APP` to your makefile
- Look in `/BlueLight/current/bglsys/Make.rules`
  - Definitions of default flags for cross-compilation
  - Definitions of compilers and tools for cross compiling
    - Gnu currently available: `CC_RTS`, `F77_RTS`, `CXX_RTS`
    - XL compilers coming: `CC_XL`, `F77_XL`, `CXX_XL`
      - May also support the remote XL compilers…
      - XLF is required for F90 codes…

DISCLAIMER

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

3D Torus Network

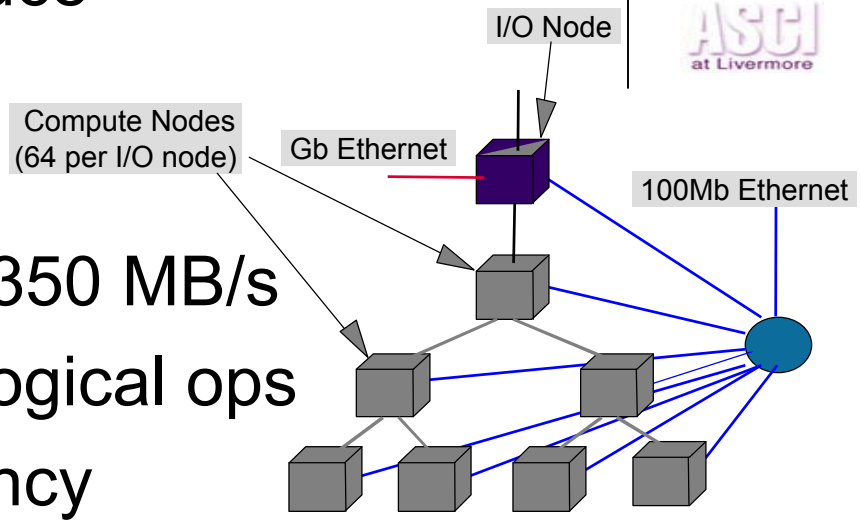




- 32 nodes × 32 nodes × 64 nodes
- Target 2 bits/cycle/link/direction
  - 175 MB/s per link per direction
  - 2.1 GB/s aggregate in & out node
  - 67.2 TB/s aggregate over all nodes
  - 358 GB/s Bisection BW
  - Node input BW/flops = .75B/flop (using 1 DHFPU)
- Both adaptive and deterministic deadlock-free routing
- Packets 32-256 bytes (31-byte overhead, 12% of 256B)
- Runs at full bandwidth (in absence of contention)
- Simulations of MPI_Alltoall show 87% efficiency

Combining Tree Network

- For broadcast, reduce, allreduce
- Any node can be root
- 256-byte packets, pipelined
- Link BW target 4 bits/cycle = 350 MB/s
- 32-bit integer arithmetic and logical ops
- Target 1.5 microsec total latency



- Floating point reductions must be done as integers
  - e.g., for sum, first find max exponent
  - or do exact arithmetic with very long ints
  - for “small” reductions, torus will be faster
- Also used for point-to-point to/from I/O node
- There is another tree for global interrupts & barriers